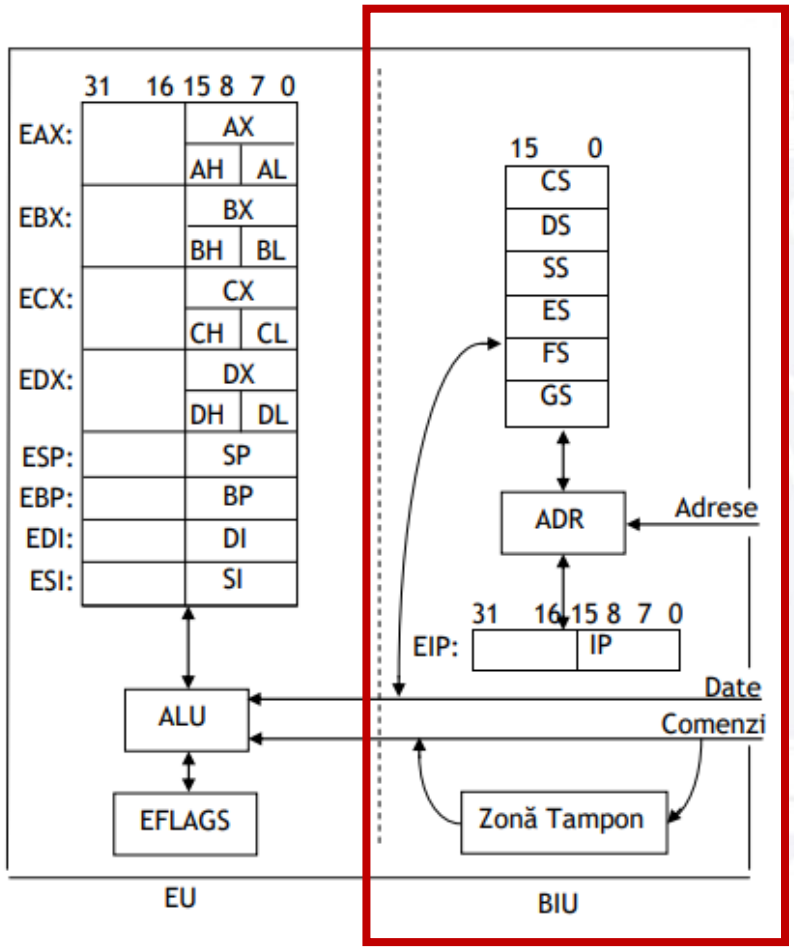
Arhitectura Sistemelor de Calcul

Lect. Dr. Șotropa Diana
diana.sotropa@ubbcluj.ro

Facultatea de Matematică și Informatică
Universitatea Babeș-Bolyai

Sistemul de adresare al Arhitecturii x86

Sistemul de adresare al Arhitecturii x86



- Nici o instrucțiune din ASM nu permite folosirea ambilor operanzi expliciți din memorie
- Componenta ADR din BIU se folosește pentru calcul de adrese
- Adresarea se face în 2 pași (de aceea există regiștrii de segment)

SEGMENTARE

- Adresa de segment
- Offset / deplasament în cadrul segmentului

Regiștrii de adresă și calculul de adresă

- **ADRESA UNEI LOCAȚII** = numărul de octeți consecutivi dintre începutul memoriei RAM și începutul locației respective
- **SEGMENT** = o succesiune continuă de locații de memorie menită să deservească scopuri similare în timpul execuției unui program

Regiștrii de adresă și calculul de adresă

CLASIFICARE:

- dpdv logic *(logical segment)*

  SEGMENT = diviziune logică a memoriei unui program, caracterizată prin **ADRESA DE BAZĂ** (început), **LIMITĂ** (dimensiune, sizeof) și **TIPUL** acesteia (cod, date, stivă, extra)

- dpdv fizic *(physical segment)*

  SEGMENT = un bloc de memorie de dimensiune fixă
  – 64 KB pentru procesoare pe 16 biți
  – 4 GB pentru procesoare pe 32 biți

Regiștrii de adresă și calculul de adresă

- **OFFSET sau DEPLASAMENT** = adresa unei locații față de începutul unui segment

Un offset e valid ⇔valoarea sa numerica pe 32 biți nu depășește LIMITA segmentului în care se raportează

- **SPECIFICARE DE ADRESĂ** = pereche formată din **SELECTOR DE SEGMENT** și un **OFFSET**

specificare de adresă = adresă segment : offset

(16 biți) (32 biți)

$$s_3s_2s_1s_0 : o_7o_6o_5o_4o_3o_2o_1o_0$$

- **SELECTOR DE SEGMENT** = valoare pe 16 biți care identifică în mod unic segmentul accesat și caracteristicile lui

= un descriptor de segment

Regiștrii de adresă și calculul de adresă

- $s_3 s_2 s_1 s_0$: $o_7 o_6 o_5 o_4 o_3 o_2 o_1 o_0$

Furnizat de SO

baza segmentului este $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$ si are limita $l_7 l_6 l_5 l_4 l_3 l_2 l_1 l_0$.

Regiștrii de adresă și calculul de adresă

- **ADRESA DE SEGMENTARE** (ADRESĂ LINIARĂ) = baza + offset

$$a_7a_6a_5a_4a_3a_2a_1a_0 = b_7b_6b_5b_4b_3b_2b_1b_0 + o_7o_6o_5o_4o_3o_2o_1o_0 .$$

E făcută de ADR, nu se poate accesa de către programatori

- Adresă FAR = adresă completă, specificare de adresă
- Adresă NEAR = adresă care se precizează doar prin offset

Regiștrii de adresă și calculul de adresă

8:1000h	Specificare de adresă

Pentru a calcula adresa liniară ce-i corespunde acestei specificări, procesorul va proceda după cum urmează:

1. **Verifică dacă segmentul ce corespunde valorii de selector 8 a fost definit de către sistemul de operare** şi se blochează accesul dacă nu a fost definit un astfel de segment;
2. **Extrage adresa de bază (B) şi limita acestui segment (L),** de exemplu, ca rezultat am putea avea B = 2000h şi L = 4000h; (este o operaţie la ale cărei detalii NU avem acces, ea derulându-se exclusiv între procesor şi SO)
3. **Verifică dacă offsetul depăşeşte limita segmentului**: 1000h > 4000h ? în caz de depăşire accesul ar fi fost blocat (memory violation error);
4. **Adună offsetul cu B**, obţinând în cazul nostru adresa liniară 3000h (1000h + 2000h). Acest calcul este efectuat de către componenta ADR din BIU.

Acest mecanism de adresare poartă numele de **segmentare**, vorbind astfel despre **modelul de adresare segmentată**.

Regiștrii de adresă și calculul de adresă

În cazul în care segmentele încep la adresa 0 şi au dimensiunea maximă posibilă (4GB), orice offset este automat valid şi segmentarea nu contribuie efectiv în calculul adreselor.

Astfel, având $\mathbf{b_7b_6b_5b_4b_3b_2b_1b_0 = 00000000}$, calculul de adresă pentru adresa logică $\mathbf{s_3s_2s_1s_0 : o_7o_6o_5o_4o_3o_2o_1o_0}$ va rezulta în adresa liniară:

$$\mathbf{a_7a_6a_5a_4a_3a_2a_1a_0 := 00000000 + o_7o_6o_5o_4o_3o_2o_1o_0}$$

$$\mathbf{a_7a_6a_5a_4a_3a_2a_1a_0 := o_7o_6o_5o_4o_3o_2o_1o_0}$$

Acest mod particular de utilizare a segmentării, folosit de către majoritatea sistemelor de operare moderne poartă numele de **model de memorie flat**.

Regiștrii de adresă și calculul de adresă

- Procesoarele x86 suportă şi un mecanism de control al accesului la memorie numit **paginare**, independent de adresarea segmentată.

- Atât calculul de adrese cât şi folosirea mecanismelor de segmentare şi paginare sunt influenţate de modul de execuţie al procesorului, procesoarele x86 suportând următoarele moduri de execuţie mai importante:
  - **mod real**, pe 16 biţi (folosind cuvânt de memorie de 16 biţi şi având memoria limitată la 1MB);
  - **mod protejat** pe 16 sau 32 biţi, caracterizat prin folosirea paginării şi segmentării;
  - **mod virtual** 8086, permite rulare programelor de tip mod real alături de cele de mod protejat;
  - **long mode**, pe 64 sau 32 biţi, unde paginarea este obligatorie in timp ce segmentarea este dezactivată.

În cadrul cursului nostru ne vom concentra asupra arhitecturii şi comportamentului procesoarelor din familia Intel x86 în **modul protejat pe 32 de biţi.**

Regiștrii de adresă și calculul de adresă

Arhitectura x86 permite folosirea a patru tipuri de segmente cu roluri diferite:

- **segment de cod**, care conţine instrucţiuni maşină;
- **segment de date**, care conţine date asupra cărora se acţionează în conformitate cu instrucţiunile;
- **segment de stivă**;
- **segment suplimentar de date** (extrasegment).

- În fiecare moment este ACTIV cel mult câte un segment din fiecare tip
- CS, DS, SS, ES – conțin valorile selectorilor segmentelor active corespunzătoare fiecărui tip
- Doar segmmentul de cod este obligatoriu!

Regiștrii de adresă și calculul de adresă

Noţiune	Reprezentare	Descriere
Specificare de adresă, adresă logică,adresă FAR	$Selector_{16}:offset_{32}$	Defineşte complet atât segmentul cât şi deplasamentul în cadrul acestuia
Selector de segment	16 biţi	Identifică unul dintre segmentele disponibile. Ca valoare numerică acesta codifică poziţia descriptorului de segment selectat în cadrul unei tabele de descriptori.
Offset, adresă NEAR	$Offset_{32}$	Defineşte doar componenta de offset (considerând segmentul cunoscut ori folosirea modelului de memorie flat)
Adresă liniară (adresă de segmentare)	32 biţi	Inceput segment + offset, reprezintă rezultatul calculului de adresă
Adresă fizică efectivă	Cel puţin 32 biţi	Rezultatul final al segmentării plus, eventual, paginării. Adresa finală obţinută de către BIU, indicând în memoria fizică (hardware)

Regiștrii de adresă și calculul de adresă

- Pentru a adresa o locaţie din memoria RAM sunt necesare două valori: una care să indice segmentul, alta care să indice offsetul în cadrul segmentului.
- Pentru a simplifica referirea la memorie, microprocesorul derivă, în lipsa unei alte specificări, adresa segmentului din **unul dintre regiştrii de segment CS, DS, SS sau ES**.
- Alegerea implicită a unui registru de segment se face după nişte reguli proprii instrucţiunii folosite.

Regiștrii de adresă și calculul de adresă

Adrese NEAR

- Prin definiţie, o adresă în care se specifică doar offsetul, urmând ca segmentul să fie preluat implicit dintr-un registru de segment poartă numele de **adresă NEAR** (adresă apropiată).
- O adresă NEAR se află întotdeauna în interiorul unuia din cele patru segmente active.

Regiștrii de adresă și calculul de adresă

Adrese FAR

- O adresă în care programatorul indică explicit un selector de segment poartă numele de **adresă FAR** (adresă îndepărtată).
- O adresă FAR este deci o SPECIFICARE COMPLETA DE ADRESĂ şi ea se poate exprima la nivelul unui program în trei moduri:
  - **$s_3s_2s_1s_0$: specificare_offset,** unde $s_3s_2s_1s_0$ este o constantă;
  - **registru_segment : specificare_offset**, registru segment fiind CS, DS, SS, ES, FS sau GS;
  - **FAR [variabilă]**, unde variabilă este de tip QWORD şi conţine cei 6 octeţi constituind adresa FAR *(ceea ce numim variabila pointer in limbajele de nivel inalt)*
- **Formatul intern al unei adrese FAR** este: la adresa mai mică se află offsetul, iar la adresa mai mare cu 4 (dublucuvântul care urmează după dublucuvântul curent) se află cuvântul ce conţine selectorul care indică segmentul.
- Reprezentarea adreselor respectă principiul **reprezentării little-endian**: partea cea mai puţin semnificativă are adresa cea mai mică, iar partea cea mai semnificativă are adresa cea mai mare.

Regiștrii de adresă și calculul de adresă

Calculul offsetului unui operand. Moduri de adresare

- În cadrul unei instrucţiuni există 3 moduri de a specifica un operand pe care aceasta îl solicită:
  - **modul registru**, dacă pe post de operand se află un registru al maşinii;

```
mov eax, 17
```

E specificat in mod registru

Regiștrii de adresă și calculul de adresă

Calculul offsetului unui operand. Moduri de adresare

- În cadrul unei instrucţiuni există 3 moduri de a specifica un operand pe care aceasta îl solicită:
  - **modul registru**, dacă pe post de operand se află un registru al maşinii;
    ```
    mov eax, 17
    ```
  - **modul imediat**, atunci când în instrucţiune se află chiar valoarea operandului (nu adresa lui şi nici un registru în care să fie conţinut);
    ```
    mov eax, 17
    ```

E specificat in mod imediat

Regiștrii de adresă și calculul de adresă

Calculul offsetului unui operand. Moduri de adresare

- În cadrul unei instrucţiuni există 3 moduri de a specifica un operand pe care aceasta îl solicită:
  - **modul registru**, dacă pe post de operand se află un registru al maşinii;
    ```
    mov eax, 17
    ```
  - **modul imediat**, atunci când în instrucţiune se află chiar valoarea operandului (nu adresa lui şi nici un registru în care să fie conţinut);
    ```
    mov eax, 17
    ```
  - **modul adresare la memorie**, dacă operandul se află efectiv undeva în memorie. În acest caz, offsetul lui se calculează după următoarea formulă:

adresa_offset = [bază] + [index × scală] + [constanta]

Regiștrii de adresă și calculul de adresă

Calculul offsetului unui operand. Moduri de adresare

adresa_offset = [bază] + [index × scală] + [constanta]

- Deci **adresa_offset** se obţine din următoarele (maxim) patru elemente:
  - conţinutul unuia dintre regiştrii EAX, EBX, ECX, EDX, EBP, ESI, EDI sau **ESP** ca **bază**;
  - conţinutul unuia dintre regiştrii EAX, EBX, ECX, EDX, EBP, ESI sau EDI drept **index**;
  - factor numeric (**scală**) pentru a înmulţi valoarea registrului index cu 1, 2, 4 sau 8 - valoarea unei constante numerice, pe octet, cuvant sau dublucuvânt.

- De aici rezultă următoarele moduri de adresare la memorie:
  - **Directă**, atunci când apare numai constanta;
  - **Indirectă**, atunci când apare unul dintre regiștrii de bază sau de index;
    - **bazată**, dacă în calcul apare unul dintre regiştrii bază;
    - **scalat-indexată**, dacă în calcul apare unul dintre regiştrii index;
- Cele trei moduri de adresare a memoriei pot fi combinate. De exemplu, poate să apară adresare directă bazată, adresare bazată şi scalat-indexată etc

Regiștrii de adresă și calculul de adresă

Calculul offsetului unui operand. Moduri de adresare

- Adresarea care NU este **directă** se numeste **adresare indirectă** (bazată şi/sau indexată). Deci o **adresare indirectă** este cea pt care avem specificat cel puţin un registru între parantezele drepte.

- La instrucţiunile de salt mai apare şi un alt tip de adresare numit **adresare relativă**.
- Adresa relativă indică poziţia următoarei instrucţiuni de executat, în raport cu poziţia curentă. Poziţia este indicată prin numărul de octeţi de cod peste care se va sări.
- Arhitectura x86 permite atât adrese relative scurte (SHORT Address), reprezentate pe octet şi având valori între -128 şi 127, cât şi adrese relative apropiate (NEAR Address), pe dublucuvânt cu valori între -2147483648 şi 2147483647.

Regiștrii de adresă și calculul de adresă

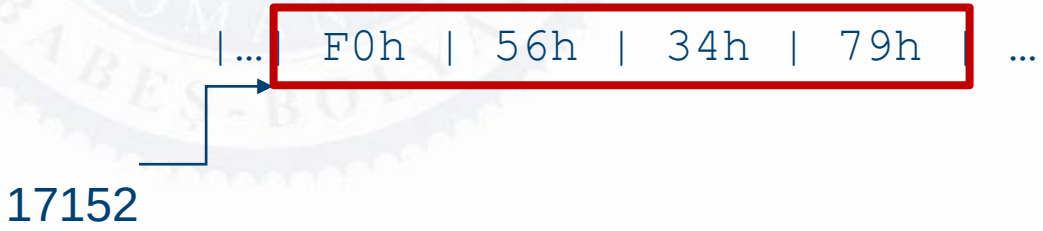
```
;EBX = 17152
MOV EAX, EBX; EAX = ?
MOV EAX, [EBX]; EAX = ?
MOV EAX, [EBX + 4*ESI - 17]


a dw 21712
…
MOV AX, [a]; AX = ?
```

Regiștrii de adresă și calculul de adresă

```
;EBX = 17152
MOV EAX, EBX; EAX = 17152 - ambii operanzi de tip registru
MOV EAX, [EBX]; EAX = ?
MOV EAX, [EBX + 4*ESI - 17]


a dw 21712
…
MOV AX, [a]; AX = ?
```



Regiștrii de adresă și calculul de adresă

```
;EBX = 17152
MOV EAX, EBX; EAX = 17152
MOV EAX, [EBX]; EAX = dublucuvântul din memoria RAM de la
adresa 17152, adică EAX = 793456F0h (! Little endian)
- op sursă este op din memorie cu adresare indirectă
- op destinație este op registru

MOV EAX, [EBX + 4*ESI - 17]


a dw 21712
…
MOV AX, [a]; AX = ?
```

Regiștrii de adresă și calculul de adresă

```
;EBX = 17152
MOV EAX, EBX; EAX = 17152
MOV EAX, [EBX]; EAX = dublucuvântul din memoria RAM de la
adresa 17152, adică EAX = 793456F0h (! Little endian)
MOV EAX, [EBX + 4*ESI - 17]


a dw 21712
…
MOV AX, [a]; AX = 21712
```

Regiștrii de adresă și calculul de adresă

```
;EBX = 17152
MOV EAX, EBX; EAX = 17152
MOV EAX, [EBX]; EAX = dublucuvântul din memoria RAM de la
adresa 17152, adică EAX = 793456F0h (! Little endian)
MOV EAX, [EBX + 4*ESI - 17]


a dw 21712
…
MOV AX, [a]; AX = 21712
```

Ce este o variabilă? Ce fel de operand este [a]? Mod registru, mod imediat, mod adresare la memorie?

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
UNIVERSITATEA BABEȘ-BOLYAI

Str. Mihail Kogălniceanu nr. 1
Cluj-Napoca, Cluj, România

www.cs.ubbcluj.ro